



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Am

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/731,678	12/06/2000	Sung-Hee Do	A0734/7001 (EJR)	9300

7590 05/19/2005

Edward J. Russavage
Wolf, Greenfield & Sacks, P.C.
600 Atlantic Avenue
Boston, MA 02210

EXAMINER

VU, TUAN A

ART UNIT	PAPER NUMBER
----------	--------------

2193

DATE MAILED: 05/19/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

09/731,678

Applicant(s)

DO ET AL.

Examiner

Tuan A. Vu

Art Unit

2193

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 11 March 2005.
- 2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1,3-15,17-51,53-64 and 66-95 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1,3-15,17-51,53-64 and 66-95 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. _____.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____.
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____.
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 11/05/2004.

As indicated in Applicant's response, no claims have been amended. Claims 1, 3-15, 17-51, 53-64, and 66-95 are pending in the office action.

In view of the submitted and accepted Request for Continued Examination (RCE) as filed 3/11/2005, the argument in a previous reply filed 11/05/2004 requesting withdrawal of the finality of a previous Office Action has now become moot. This Office Action herein is addressing the merits of the resubmitted claims as well as Applicants' arguments as per the above-mentioned RCE.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

3. Claims 1, 3-5, 64 and 66 are rejected under 35 U.S.C. 103(a) as being unpatentable over Richburg, USPN: 5,159,687 (hereinafter Richburg) in view of Kim et al., "Design of Software Systems based on Axiomatic Design", Robotics & Computer-Integrated Manufacturing, Vol. 8., MIT, 1991, pp. 243-255 (hereinafter Kim).

As per claim 1, Richburg discloses a digital information product comprising: a computer-readable medium and, stored therein, computer program instructions defining a software system that produces code based on a set of requirements and design specifications, or parameters by the programmer (e.g. col. 3, lines 23-54; col. 7, lines 37-50; *variables* – col. 5,

Art Unit: 2193

lines 38-51; *knowledgeable, applications requirements* -- col. 10, line 41 to col. 11, line 24; col. 18, line 61 to col. 19, line 20; *Order Processing/user defined code* -Fig. 2; Fig. 3, 4; *specific variables or requirements* - col. 19, lines 3-9; *knowledgeable database, expert system code generator* -- col. 20, line 42 to col. 21, line 22).

But Richburg does not explicitly specify that the requirements are functional requirements but suggests accommodating target application's functionality when generating software code based on requirements database and CASE tool to analyze requirements (e.g. col. 6, line 49 to col. 7, line 61; col. 10, lines 33-40). The concept of translating functional requirements from customer needs and fulfilling them with implementation in a software or other designs using CASE tool/middleware (e.g. Rationale Rose/UML) was a known concept at the time the invention was made; and establishing or mapping those functional requirements against the design specification or parameters of a target application was also a known concept; hence, the disclosing by Richburg that requirements on functional aspects of an application of a certain genus are being mapped against a knowledgeable base to generate code (e.g. col. 3, line 29 to col. 4, line 4), i.e. using functional requirements as base to specify how a program is to be implemented to require desired functions, is implicitly disclosed.

Nor does Richburg explicitly disclose that program instructions defining the software system, well executed, allow the programmer to define a design matrix describing a relation between the set of functional requirements (FR) and the design parameters (DP). The concept of mapping requirements against all the functionalities of a target software application, e.g. validation spreadsheet or traceability matrix in Modeling tool, was a known concept at the time the invention was made; and Richburg discloses a framework using Case tool wherein

Art Unit: 2193

requirements from a customer's needs are mapped into requirements in terms of software objects created. Kim, in a methodology to generate the best implementation of code based on consumer or process domain requirements, discloses a software framework that executes a mapping of functional requirements to design parameters and code generation analogous to Richburg's method and the use of matrix to describe such mapping relation (e.g. *FR, DP* - pg. 246-248, ch. *Hierarchical structuring and decomposition*). In view of the well-known concept in mapping requirements against design specifications in software design framework, it would be obvious for a skill in the art at the time the invention was made to add to Richburg's software product instructions to perform the mapping of functional requirements against design parameters with matrix as taught by Kim. The motivation would be obvious since having included in the framework program capability to provide a visual structure like matrix to verify the correctness of what to implement versus predetermined functional requirements would enhance considerably the stages of designing, implementing, and verifying software product.

As per claim 64, Richburg discloses a database format for designing a software system, comprising:

design identification information for identifying information describing the software system (e.g. *program requirements* – col. 3, lines 42-54; *case design 12a, 12b* - Fig. 1); and

detailed design description information (e.g. Fig. 2; col. 27, line 22 to col. 28, line 62; *design details* – col. 18, lines 28-58 – Note: Knowledgeable files and ISF and ICF and/or header files data are equivalent to code generating information necessary to build target software system application) and software code information (e.g. col. 25, line 5 to col. 29, line 5).

But Richburg does not explicitly specify detailed design description information is for describing the structure and operating qualities of the software system. However, Richburg discloses Case tools to capture all functionalities required for the target application (e.g. Fig. 1); then Richburg discloses table information for describing program language structure of the code instructions (e.g. col. 25, line 5 to col. 27, line 11) and header files from the *knowledgeable* database (col. 27, line 22 to col. 28, line 62). In other words, Richburg provides the definition of operational and structural details by means of user definition as in Case tools (Note: Case tool inherently define operations interaction between structural objects defined by the user) so to assemble requirements and engineering language into a knowledgebase (col. 10, lines 28-50; col. 7, lines 2-53; *Master Knowledgeable file* - col. 22, line 15 to col. 23, line 9). All of these above amount to disclosing a detailed output form to incorporate all description of structure and operating qualities of the software to be built; hence Richburg has disclosed detailed description of structure and operating qualities.

Nor does Richburg disclose that the detailed design description defines a design matrix describing a relation between a plurality of functional requirements (FR) and design parameters (DP), represents one object by at least one FR and represents one object by at least one DP. But in view of the teachings by Kim to represent software domains in terms of subroutines, operating systems, of compilers and hierarchy of elements thereof (pg. 244, right column) and mapping functional domain to physical domains (*FR, DP* - pg. 246-248), it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide Richburg's requirement mapping engine with the teachings by Kim so that one or more object is representing at least some FR or DP, and performing the matrix to implement such mapping for

Art Unit: 2193

the same reasons as set forth in claim 1 above; and describe Kim's matrix mapping in a detailed design document as established in most software development tool, the likes of which evidenced via Richburg's specification detail files in conjunction with the use of Case tools data being persisted in the knowledgebase as set forth above; thereby providing legacy of stored information enabling future reuse and customizing of application as intended by Richburg.

As per claims 3-5, Richburg in combination with Kim provides program instructions to allow the developer to define a matrix describing relation between functional requirements (FR) and design parameters (DP) and this limitation has been addressed in claim 1. But Richburg does not specify that these instructions allow a programmer to manipulate said matrix into lower triangular form (re claim 3), to determine a decoupled design matrix (re claim 4), or to determine an uncoupled design matrix (re claim 5). In extension to the teachings by Richburg in using rules base inference and expert system for solving complex operations so as to provide a reusable knowledgebase (Richburg: Appendix, col. 18-33), Kim discloses a mathematical approach similar to the expert rule-based engine by Richburg, to allow the developer to define such matrix to be manipulated as in coupled form, in a lower triangular corner or decoupled form; or a uncoupled form as claimed (e.g. *Coupled design*- equ. 9, *Decoupled design*- equ. 7,8, *Uncoupled design*- equ. 10--) in order to select the best design based on some axiomatic information. It would have been obvious for one of ordinary skill in the art at the time the invention was made to use the axiomatic approach by Kim (*coupled, decoupled, uncoupled design*) when generating structures for mapping requirements defined from the Case tools, and when utilizing expert inference engine, and knowledgebase as taught by Richburg because according to Kim, this approach would define the most acceptable software systems for

Art Unit: 2193

implementing wide range of requirements (as suggested by Richburg in the case tool and reusable expert database for different applications); and also rapid adaptation to large number of domain ideas and selection of the best solutions for acceptable options.

As per claim 66, see Richburg (e.g. Fig. 2; col. 27, line 22 to col. 28, line 62).

4. Claims 6-13, 15, 17-51, 53-63, 67-94 are rejected under 35 U.S.C. 103(a) as being unpatentable over Richburg, USPN: 5,159,687, and Underwood, USPN: 6,523,027, and further in view of Rudolph, "On a Mathematical Foundation of axiomatic design", 8/22/1996, ASME Design Engineering Technical Conference and Computers in Engineering Conference (hereinafter Rudolph).

As per claim 6, Richburg discloses a method for producing software system, such method producing code based on a set of requirements and design specifications, or parameters (re claim 1 for rejection) and implicitly discloses that such set of requirements are functional requirements. Richburg further discloses using rules base inference and expert system for solving complex operations so as to provide a reusable knowledgebase (Richburg: Appendix, col. 18-33). The concept of mapping requirements against all the functionalities of a target software application or hardware application, e.g. validation spreadsheets or traceability matrices, was a known methodology in a variety of design and engineering development at the time the invention was made; and Richburg is but one of other instances thereof using Case tool (col. 6, line 49 to col. 7, line 61). As an extension to Richburg inference engine and complex problem solving and requirements mapping, Underwood, for example, discloses using a matrix to evaluate Firewall implementation against system requirements (e.g. col. 289, lines 1-13; Fig. 119) so as to determine the most appropriate design to match a given set of requirements. In

Art Unit: 2193

another domain of design engineering, for example, Rudolph also provides mathematical approach to map requirements against design parameters using matrices (pgs. 2-4, ch. 2). It would have been obvious for one of ordinary skill in the art at the time the invention was made to implement to matrix methodology as suggested by Underwood or Rudolph to complement to the requirements mapping performed by Richburg's expert inference system because this matrix approach, like in many applications, provide efficient visualization of mapping operations to a wide variety of expert domain data to many genus of applications in as many level of details as given by matrix definition, just as has been used in well-known software means of requirements validation or verification.

As per claim 7, in reference to claim 1 above, Richburg combined with Underwood does not specify a step of constructing functions using diagonal elements of the design matrix. The correspondence of requirements or FR items to more than one design parameters (DP), e.g. a one-to-one or one-to-many correspondence between FR and DP, are known relationships in object-oriented modeling and requirements construction in Case tool like Rational Rose at the time the invention was made. And instances in which only diagonal elements describe the functionality of the software to be built is but one relationship in matrix mapping so as to validate 'what it is versus what it does, how it is controlled' according to Richburg (col. 21, lines 31-33) in a framework using Case tool as mentioned above. As mentioned above, any requirement mapping using matrix is to find out what is the best solution given a set of requirements and design parameters en route to customizing applications for solving real-world problems. Rudolph, as mentioned above, discloses using a known concept which is a matrix to map FR to DP as claimed, to allow the developer to define such matrix to be manipulated in a

Art Unit: 2193

lower triangular corner or decoupled form; to be defined in a uncoupled form or diagonale mapping as claimed (e.g. *Coupled design*, *Decoupled design*, *Uncoupled design* -- pgs. 2-4, ch. 2) in order to select the best design based on some axiomatic information. It would have been obvious for one of ordinary skill in the art at the time the invention was made to implement to the mapping thus disclosed by Underwood/Richburg so that diagonale mapping, as in a one-to-one or one-to-many mapping being but one instance among other design instances as mentioned above and further exemplified in the FR/DP relationship matrix as mentioned by Rudolph, because this manipulation of the matrix into some desired patterns (e.g. coupled, uncoupled or diagonale mapping, or decoupled) for matching the requirements to the design resources would allow visualization of what is best for what has been required or needed as contemplated by both Richburg and Rudolph.

As per claim 8, Richburg discloses using Case tools for processing of code structures into generating of code files (e.g. col. 5, line 52 to col. 6, 33). Likewise, Underwood utilizes Case tools to model the design objects used to implement the required specifications of the program logic (*Rational Rose* – Fig. 36, 122; col. 97, lines 24-62) and further describe the relationships between modules with entity-relation diagram (e.g. col. 96, lines 4-28) and activity of object interactions at runtime (e.g. Fig. 19, 20A), all of which being a representation of software objects or modules with interrelations equivalent to flow diagram of a software system. It would have been obvious for one of ordinary skill in the art at the time the invention was made to add to the Case tools of Richburg the case tools modeling and diagram flow representation as taught by Underwood in the process of generating code. The motivation is that the modeling with representation of system's components inter-dependency or flow is helpful in determining

Art Unit: 2193

the relationships between code object or modules and thereby enabling an object-oriented design with all the benefits from the OOD modeling and modularization and reusability therefrom, as well as a better correlating of FR and DP derived from such modular representations.

As per claim 9-13, As to extend Richburg's suggested use of Case tool to perform requirements gathering, Underwood discloses modeling via Case Tools such as Rational Rose, ERD, and Java ASP pages modeling (e.g. Fig. 18, 122, 47; col. 96, lines 4-28; col. 99 line 35 to col. 100, line 33) and has implicitly disclosed defining a software module corresponding to one functional requirement (re claim 9 – Note: each object in an Use case depict a functional requirement) ; defining a software module based upon a flow description (re claim 10); defining a plurality of software modules corresponding each to a FR (re claim 11); relating (re claim 12) the plurality of software modules by junctions (see Fig. 19, 122); and (re claim 13) combining such modules according to their junctions (Note: this combination of object modules linked by relationships is inherent to the modeling and code generation process in Rational Rose development). The motivation to modify Richburg's case tools so to implement the code using modeling with system diagram flow as taught by Underwood, and modularizing process thereby would be obvious herein using the rationale as set forth in claim 8 above.

As per claim 15, this claim can be rejected using the rejection of claim 6.

As per claims 17-19, these claims recite the uncoupled matrix, diagonal form matrix, and decoupled matrix, all of which limitations are disclosed in the combination using Rudolph as set forth in claim 6; and are rejected herein using the same rationale.

As per claim 20, Richburg does not disclose generating a plurality of models and outputting of models for a code generator but discloses a plurality of files describing code

Art Unit: 2193

specifications to be submitted to the code processor (e.g. Fig. 2-3). Using the teachings by Underwood in claim 8 (and again in claims 9-13) along with the Case Tools utilization by both Richburg and Underwood, this limitation would have been obvious for the same reasons as set forth therein.

As per claims 21-23, Richburg discloses case Tools and object reusability type of approach (e.g. *standardized software units, extensibility* - col. 3, line 49 to col. 4, line 16; col. 8, lines 1-17) and a programming language that is class-based (e.g. col. 12, lines 54-60) but does not explicitly specify generating (re claim 21) a diagram of a object-oriented structure, (re claim 22) a diagram in a UML format, or (re claim 23) a group of diagrams comprising use case and E-R diagram. But these limitations are implicitly disclosed by the teachings of Underwood as mentioned in claims 8-13 from above. These limitations would also been obvious in view of the rationale used therein, i.e. the motivation for modifying the class-based programming language and case tools and reusability approach by Richburg with the Rationale Rose case tools (i.e. object-oriented, UML format) as suggested by Underwood for modeling, or with E-R diagrams would have been obvious for the same reasons as mentioned in the rejections of claims 8-13.

As per claim 24, only Underwood discloses generating object-oriented entities (e.g. Fig. 18, 122, 47; col. 96, lines 4-28; col. 99 line 35 to col. 100, line 33; Fig. 54-56) in view of the teachings of using Object-Oriented case tools to model the business logic and generating code (re claim 8). The motivation to modify Richburg's teachings on case tools to implement the object-oriented modeling and diagramming for generating OO code would be obvious using the same rationale as used in claims 8-13 above.

As per claim 25, Richburg discloses case Tools and a programming language that is class-based (e.g. col. 12, lines 54-60) but it is Underwood who discloses defining OOP classes to fulfill a requirements (e.g. *implement ... requirements and design* - col. 13, line 10 to col. 14, line 26). The fact of using Rational Rose case tools to model class entities is equivalent to customizing requirements per object instances or class entity. The combination with Richburg's case tools for generating code would be obvious for the same rationale mentioned in claims 21-23 above, and because of the benefits pertinent to developing in an object-oriented approach such as reusability, portability, modularization, extensibility, polymorphism, encapsulation, etc.

As per claims 26-27, since classes are known to be organized in a hierarchy of grand-parents, parents, and children, the limitations of defining a child, grand-child of a class using a particular FR would also been obvious by virtue of the above rejection.

As per claims 28-29, these claims would also been obvious for the same rationale used in claim 25-27 above.

As per claim 30, Richburg (in combination with Underwood/Rudolph) discloses generating source code (e.g. *Program file 16* – Fig. 2).

As per claim 31, Richburg discloses a method for designing software, such method comprising: defining a relation between a plurality of requirements of the software system and design parameters (e.g. col. 3, lines 23-54; col. 7, lines 37-50; col. 18, line 61 to col. 19, line 20; *variables* – col. 5, lines 38-51; *knowledgeable, applications requirements* -- col. 10, line 41 to col. 11, line 24; Fig. 3, 4; *specific variables or requirements* - col. 19, lines 3-9; *knowledgeable database, expert system code generator* -- col. 20, line 42 to col. 21, line 22).

Art Unit: 2193

But Richburg does not explicitly specify that the plural requirements are functional requirements; but this limitation has been addressed in claim 1 above using Underwood.

Nor does Richburg explicitly disclose representing at least one object-oriented object by at least one of the functional requirements (FR), and by at least one design parameters (DP); but this limitation has been addressed in claim 9 above.

Nor does Richburg disclose explicitly that the software design involves object-oriented objects. In view of Richburg's approach suggesting reusable of units of code and Case tool (*standardized software units, extensibility* - col. 3, line 49 to col. 4, line 16; col. 6, line 49 to col. 7, line 61), such reminiscent of object-oriented extensibility and modularization propriety (e.g. col. 2, lines 26-37) as afore-mentioned in claim 21, this limitation is implicitly disclosed, or if not would have been obvious for the same reasons used in claim 21.

Nor does Richburg teach defining a design matrix to correlate a plurality of FRs and DPs. This limitation has been addressed in claim 6 above; hence is rejected herein likewise.

Nor does Richburg disclose representing a method of at least one object-oriented software object by a product of a portion of the matrix and at least on DP. But in view of the mathematical approach by Rudolph using a matrix multiplication scheme to derive an matrix association or product element (or method as claimed) by matching FR and DR matrices (e.g. pg. 2, ch. 2: math expressions (1) and (2)) as used in claims 2-4 rejection, this limitation would also been obvious for the same rationale.

As per claim 32, Richburg in combination with Underwood disclose defining in a database functional requirements and system constraints (re claim 31).

As per claims 33-35, only Rudolph teaches the matrix approach to equate a set of FRs with a product by or association with set of DPs (see Rudolph: pg. 2-4, ch.2); the organizing of FRs into a leaf elements, or DPs into leaf elements (rows or columns) of the matrix, or process variable into a leaf would have been thereby suggested, the process variable (PV) or design parameters being interchangeable sets to be mapped against the FR set in order to correlate program specification to FR as suggested by Richburg (e.g. *variables* – col. 5, lines 38-51; *knowledgeable statements, parameters substitution* - col. 10, line 47 to col. 11, line 62). As has been obvious according to the corresponding rejection of claim 31, the usage of the matrix to determine correlation or method between program variables or design parameters as taught by Richburg and further enhanced by Underwood and Rudolph, the limitation to set FR and DP or PV as leaf elements in the row or columns of the matrix as suggested by Rudolph would also been obvious for the same rationale used in claim 31 above.

As per claim 36, Richburg discloses a software designing method comprising: defining a software system by defining a relation between FRs of the system and DPs implementing the system (re claim 31 for corresponding rejection); but does not specify defining an object-oriented object. This last limitation has been addressed in claim 31 above using Underwood's teachings.

Nor does Richburg (combined with Underwood) disclose defining a design matrix for correlating the FRs and the DPs; nor does Richburg teach defining a method that may define on a OO object, such OO object and such method are related to the design matrix. These limitations have also been addressed in claim 31 above.

As per claim 37-40, both Richburg and Underwood disclose mapping of requirements against OO program specifications or parameters but do not teach such mapping by using a design matrix to map FRs and DPs (such limitation been addressed above using Rudolph); nor does Richburg (with Underwood's teachings) disclose object class, its instance, its behavior against 1st level, 2nd level, or 3rd level of FR, respectively and all those levels against a hierarchy of FRs (re claim 37, 38, 39, 40) . Rudolph further discloses such mapping being applied to hierarchy of objects in accordance with nth level rank matrix mapping (e.g. pg. 4, Fig. 3; pg. 4-5, ch. 3) for evaluation of the most optimal design choice as earlier suggested by Underwood (col. 289, lines 1-13; Fig. 119) in claim 31 above. It would have been obvious for one of ordinary skill in the art at the time the invention was made to modify the OO approach by Richburg (combined with Underwood) to map FR and program objects so that a hierarchy of objects, such as class, class instance or class behavior are matched via the matrix of FRs organized into multi-rank levels as suggested by the design evaluation by Rudolph. The motivation would be obvious because this would enable Richburg's method to address each and every level of hierarchy of objects making up the object-oriented approach code implementation suggested by Richburg and furthered by Underwood, thereby enabling a more accurate validation of every aspects of the OO assembling of elements in accordance to the level of subdivision imparted to the corresponding FRs as mentioned by Rudolph's matrix-based evaluation.

As per claim 41, in view of the teachings by Rudolph to use design matrix and the combination as mentioned in claim 36, the mapping of FR into the implementation domain via using DPs would also been obvious for the same grounds as set forth therein.

As per claims 42-43, refer to claims 7 and 5, respectively.

As per claims 44-47, these claims correspond to claims 20, 21, 23 and 22 respectively, hence are rejected using the corresponding rejections therein.

As per claim 48, Richburg discloses a database format for designing a software system, comprising: a software design specifications wherein is defined a relation between a plurality of requirements and design parameters (e.g. col. 3, lines 23-54; col. 7, lines 37-50; col. 18, line 61 to col. 19, line 20; *variables* – col. 5, lines 38-51; *knowledgeable, applications requirements* -- col. 10, line 41 to col. 11, line 24; Fig. 3, 4; *specific variables or requirements* - col. 19, lines 3-9; *knowledgeable database, expert system code generator* -- col. 20, line 42 to col. 21, line 22) and a software code produced by the design specifications (Fig. 2; Fig. 3, 4 – Note: the storing of software code for future reuse or testing purposes in a database is inherent).

But Richburg does not explicitly specify that such plurality of requirements is one of functional requirements. This limitation has been addressed in claim 1 using Underwood.

Nor does Richburg disclose defining a design matrix to describe a relation between the FRs and the DPs. This limitation has also been addressed in claim 31 above.

Nor does Richburg disclose design specification for at least one software object being represented by at least one or more FRs, or specification for data used by such object being represented by at least one or more DPs. But this limitation would have been obvious in view of the object-oriented teachings by the combination Richburg/Underwood as being addressed in claims 37-41.

As per claim 49, Richburg discloses identifying the software code in the database (e.g. col. 7, lines 2-53; Fig. 2).

As per claim 50, Richburg discloses specifications stored as knowledgeable files in a database to correspond with the requirements database (re claim 1) but does not disclose storing therein of detailed design description including the design matrix. But in view of the combination using Rudolph to implement a matrix to correlate Richburg FR and DP (with Underwood's teachings) as seen in claim 2, this limitation would also been obvious. One of ordinary skill in the art would be motivated to provide database storage of design description including such matrix implementing such as taught by Rudolph for the same reason to support the mapping of Richburg's system just as set forth in claim 31 above, i.e. this manipulation of the matrix into some desired patterns (e.g. coupled, uncoupled, or decoupled) for matching the requirements to the design resources would further enhance the optimization of resources while fulfilling the functional requirements as implemented by the design.

As per claim 51, this claim would also have been obvious for the rationale to include a design matrix as set forth in claim 50 into the database storing the FR and the DPs as taught by Richburg would also be used herein.

As per claim 53, this claim corresponds to the organization of software design parameters or objects specifications into levels or hierarchy analogous to the level of mapping of object-oriented objects to level of FRs as discussed in claims 37-40 above; hence is rejected herein using the rationale as set forth therein. As for the referencing of such plurality of levels of DPs by another software system as recited in the instant claim, the object-oriented approach as addressed using Underwood to enhance Richburg as set forth in claims 36-40, would have implicitly disclose this limitation, given the hierarchy of class organization and method calling structure inherent to an OO programming design.

As per claim 54, see Richburg (Fig. 1-3; col. 3, lines 49-65; col. 7, lines 37-53).

As per claim 55, see Richburg discloses user interface and database, and searching therein to obtain elements to create a new software system (e.g. col. 3, lines 49-65; Fig. 2a-10; col. 6, lines 34-48).

As per claim 56, Richburg discloses the interaction of new software and several elements of the software system (e.g. col. 18, line 61 to col. 19, line 34) but Richburg does not disclose verifying if the interface is operable. Underwood, in a analogous method using user interface to specify the creation of a new software system disclose the checking consistency of data gathered during a session of user interfacing to support the fulfillment of a service request for application code (e.g. Fig. 15B, 18A, 20, 153-154). It would have been obvious for one of ordinary skill in the art at the time the invention was made to implement the user interface of Richburg so to incorporate the correctness checking of the activities in the user interface during the creation of the new software system code as suggested by Underwood because this would enhance the optimization of resources usage by obviating potential failures due to interface data retrieval incompatibilities or user-system communications data inconsistency problems as mentioned by Underwood.

As per claim 57, Richburg mentions about consistency of database to support a large scale of applications and user interfacing (e.g. col. 8, lines 1-17) and control checking for correctness in program generating based on the information of the database (e.g. col. 23, line 61 to col. 24, line 23); hence has disclosed consistency of design.

As per claim 58, see Richburg (e.g. col. 18, line 61 to col. 19, line 20; col. 25, line 4 to col. 27, line 11 – Note: operands and mnemonics --or keywords--, and comments of SEL

language stored in knowledgeable files and Application database are equivalent to design parameter information stored in database).

As per claim 59, Richburg discloses an executable form (e.g. col. 30, lines 57-62 – Note: assembly language is implicitly disclosing that source code is executable by a machine level).

As per claim 60, Richburg discloses source code (e.g. Fig. 2).

As per claim 61, Richburg disclose control checking of code constructs against knowledgeable in the database (re claim 57) and mapping of requirements with code specifications via a code processing unit (Fig. 2-5); but does not specify consistency checking of implemented design between some level of DP. But in view of the checking steps as shown in the Fig. 2-4, from high module level to low level instruction constructs of an implemented form of the program specifications, this limitation is disclosed.

As per claim 62, Richburg suggested reusable framework (e.g. col. 24, line 38 to col. 25, line 2) and Underwood teaches object-oriented design (re claims 21-24). But Richburg does not specify consistency checking of level of design so that the second level is child of first level, even though Richburg teaches about class in implementation of code (col. 12, lines 54-60). In view of the rationale to combine Underwood's approach to use object-oriented approach to implement the requirements mapping and code generation of Richburg as set forth in claim 21-24 above, it would have been obvious for one of ordinary skill in the art at the time the invention was made to modify the code checking as suggested by Richburg from above so that it does consistency checking between hierarchy of classes, e.g. child-parent checking, as suggested by Underwood from above because of the same motivations as mentioned in claims 21-24.

As per claim 63, the rationale used in claim 56 is herein used to address this instant claim.

As per claim 67, this claim includes defining a design matrix and generating code based thereon as in claim 15, hence is rejected using the corresponding rejection as set forth therein.

As per claim 68, refer to claim 32.

As per claims 69-71, refer to corresponding rejections of claims 33-35, respectively.

As per claims 72 and 74, refer to rejections of claims 22 or 47.

As per claim 73, see claim 59.

As per claim 75, Richburg does not disclose diagram describing a software code, but according to the modeling approach by Underwood as set forth in the rejection of claims 21-24, the limitation such as to describe software with a diagram such as a class diagram, interaction diagram, collaboration diagram, sequence diagram, state diagram, activity diagram as claimed would have been implicitly disclosed by the modeling with Case tools (e.g. Rational Rose; Fig. 18, 122, 47; col. 96, lines 4-28; col. 99 line 35 to col. 100, line 33; Fig. 54-56) as taught by Underwood; and would have been obvious for the same reasons used in claims 21-24.

As per claim 76, the limitation to associate at least one object/element of the diagram modeling by Underwood to the design matrix would have been obvious by virtue of the rationale using Underwood's validation of requirements (e.g. col. 289, lines 1-13; Fig. 119) and Richburg's relating of requirements to software specification as set forth in claim 2.

As per claim 77, use-case is implicitly disclosed in Underwood's teachings from claim 75, hence the limitation of using use-case based on the design matrix would have been obvious by virtue of the above rationale from claim 76.

As per claim 78, Richburg, combined with Underwood, discloses validating user's needs or requirements to conform to requirements but does not disclose hierarchy of user's needs associated with layers of use-case diagram. By virtue of the rationale as set forth in claims 37-41, using the hierarchy and multi-level structuring of the matrix elements by Rudolph and the modeling and user's requirements mapping by Underwood/Richburg, this above limitation would also be obvious using the same grounds as set forth in claims 37-41, because organizing objects or elements of an use-case is equivalent to organizing object-oriented elements by Underwood in the mapping against the multi-levels of requirements as taught Rudolph's matrix, especially when use-case is fundamentally a form of modeling instance for customizing or fulfilling a user's requirements.

As per claim 79, since the layers such as relation layer, use case layer, and actor layer are all implicitly disclosed in the case tools by Underwood, this claim is rejected with the same rationale as set forth in claim 77.

As per claim 80-82, since actor layer representing more than one actor is implicit to the Rational Rose taught by Underwood, and that a use case describing a user needs is inherent in such Case tools, or that a relation layer describes a relation between customer's needs, these claims would have been obvious with the same rationale used in claim 77.

As per claims 83-85, these claims are implicitly disclosed because most modeling sets of data are stored in some files in the development tools such as exemplified by the Rational Rose Case tools as taught by Underwood (e.g. Object Modeling -col. 97-99). In combination with the rationale as used in claims 21-23, or claims 76-77, these claims are herein rejected for being obvious over the combination and rationale as set forth therein.

As per claim 86, Richburg discloses a computer product with a program to performs a method for rendering an user interface, such method comprising displaying a software design interface (e.g. col. 4, lines 32-41; *edit pull-down menu* - Fig. 2; col. 5, line 60 to col. 7, line 16) upon which a software design is based and described in terms of correlating a set of requirements and design parameters (DP) implementing the software design (e.g. col. 3, lines 23-54; col. 7, lines 37-50; *variables* – col. 5, lines 38-51; *knowledgeable, applications requirements* -- col. 10, line 41 to col. 11, line 24; col. 18, line 61 to col. 19, line 20; *Order Processing/user defined code* -Fig. 2; Fig. 3, 4; *specific variables or requirements* - col. 19, lines 3-9; *knowledgeable database, expert system code generator* -- col. 20, line 42 to col. 21, line 22).

But Richburg does not specify that the requirements associated with the DPs are functional requirements (FR). But this limitation has been addressed in claim 1.

Nor does Richburg disclose displaying a set of interface requirements upon which the software design is based, nor does Richburg disclose displaying a design matrix for correlating said FRs to the DPs. However, Richburg teaches use of Case tool and suggests mapping for verification on requirements and design (re claim 6; col. 21, lines 31-33). The limitation for displaying a set of interface requirements would have been obvious in view of the combined teachings by Richburg/Underwood and Rudolph as set forth in claim 6 or 31.

As per claim 87, using the teachings of Rudolph as used in claims 2-5 (e.g. *Coupled design, Decoupled design, Uncoupled design* -- pgs. 2-4, ch. 2), the limitations of a decoupled, uncoupled, or coupled design would have been obvious in view to such rationale set forth therein.

As per claims 88-89, these claims would have been obvious in view of the combination as set forth in the rejection of claims 17-19 from above.

As per claims 90-91, the teachings by Rudolph disclose defining or indicating an accuracy of design with respect to an optimum design (e.g. *best design* - pg. 3, right column, 1st para.) ; or defining/indicating a range of acceptable inputs (e.g. *upper and lower limits* - pg. 5, ch. 3.1) operable by the design matrix are disclosed, rendering the instant limitations obvious according to the rationale of claims 17-19 above.

As per claims 92-93, these claims would also been obvious in view of the rationale in rejections of claims 37-41.

As per claim 94, by choosing the best design using the matrix mathematics method, Rudolph discloses the robustness of the design choice; hence the combination using Rudolph, and Richburg/Underwood as set forth in claims claims 17-19 would also render the instant limitation obvious for the same rationale therein.

5. Claim 14 is rejected under 35 U.S.C. 103(a) as being unpatentable over Richburg, USPN: 5,159,687, and Underwood, USPN: 6,523,027, in view of Rudolph, "On a Mathematical Foundation of axiomatic design", 1996; as applied to claim 13, and further in view of Kim et al., "Design of Software Systems based on Axiomatic Design".

As per claim 14, Official notice is taken that providing module flow chart in a software design and modeling, or requirements analysis was a known concept at the time the invention was made. Richburg in combination of Underwood discloses object-oriented collaborations among objects being joined (or summation) or linked together by interactions in the modeling interface inherent to Case tools. As to extend the collaboration of created objects as in a flow

Art Unit: 2193

reminiscent of case tool modeling, and the concept of using axiomatic matrix mapping method by Rudolph, Kim discloses a flow where junctions are created for summations of modules, control, and feedback (pg. 252, Fig. 10). In case the collaboration of modules in case tool by Richburg/Underwood does not already include a data flow diagram with feedback and control, it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide a module flow chart as taught by Kim to the framework of Richburg (combined with Underwood/Rudolph) so to further enhance the module interdependencies with collaborations like summation, control, and feedback association as mentioned above. The motivation for using Kim's suggestion would be to enable better understanding of dependencies of data flow and intervening controls thereof, and better evaluation on multi-directional dynamics of requirements from interacting modules used in modeling as intended by Richburg/Underwood and because of the well-known benefits derived from such data flow chart.

6. Claim 95 is rejected under 35 U.S.C. 103(a) as being unpatentable over Richburg, USPN: 5,159,687, and Underwood, USPN: 6,523,027, in view of Rudolph, "On a Mathematical Foundation of axiomatic design", 1996; as applied to claim 92, and further in view of Crampton et al., USPN: 6,415,196 (hereinafter Crampton).

As per claim 95, Richburg discloses rendering of a software code based on requirements association with database of *knowledgeable* and specifications provided by users; but does not teach arrangement of design parameters into a Gantt chart depicting production dependencies of the software design. Underwood, in an analogous system to fulfill user's requirements or specifications, discloses delivery and versioning of production code release and development management (e.g. *project management* – Fig. 43; production steps 2504. 2510 -- Fig. 25; *PVCS*

Art Unit: 2193

–, 30-31; col. 89, line 16 to col. 90, line 32). Further, Crampton, in a method to manage the development process using modeling and requirement fulfilling (see Crampton: Fig. 7, 10) like Richburg's method and analogous to the management of software development by Underwood, discloses the use of Gantt's charts to depict production dependencies and evolution of software entities or resources over time (e.g. col. 31, lines 37-63). It would have been obvious for one of ordinary skill in the art at the time the invention was made to add to the software production and delivery management as suggested by Underwood as in the combination Richburg/Underwood from claim 92 from above the implementation using Gantt chart as taught by Crampton, because this graphical depiction of a timeline evolution of resources would enhance the planning and scheduling of resources when producing the software deliverables as intended so as to support more efficiently the use of system resources in a life cycle of the program development.

Response to Arguments

7. Applicant's arguments filed 11/05/2004 have been fully considered but many of them when applicable are not persuasive. Following are the corresponding replies thereto.

(A) As for the arguments on improper combination of Richburg/Kim (Appl. Rmrks, pg. 3-4):

Applicants have submitted that Richburg relates to manipulation of solutions that have already been created/stored in the knowledgebase and converting that base into code like using a recipe (Appl Rmrks, pg. 3 middle); and such is different from the design matrix approach by Kim. In response, it is noted that Richburg knowledgebase is for storing more than just preexisting solutions; because this repository can store reusable code/scripts, the rules or methods to edit/modify these code; a storage for a engineering programming language used to actualize target code; and also is used after the Case tool to gather requirements has taken place (

Art Unit: 2193

see col. 9 line 42 to col. 10, line 60); in short a database of code objects and configuration data analogous to a reuse repository used in common Case tools based software development. Hence, there is no just massaging or re-manipulation of existing solutions as asserted, because Richburg's Case Tool analysis first provides the requirements gathering; this in turn leads to using files fetched from the knowledgebase along with the scripting rules, and if necessary the reediting of code or specifications related to the requirements; i.e. previous solutions and/or script modification or code assembling being created dynamically in conjunction with -- or as a result to -- the requirements analysis provided by Case tools. Readjusting reuse code to accommodate requirements is standard in object-oriented software development when the Case tool is used to analyze the requirements; and the mapping by Kim is not viewed a process necessarily excluded during the endeavor wherein Richburg is using results from a Case Tool session to customize the reuse components. Hence, the Applicants' assertion that using such knowledgebase like 'a recipe' is far from convincing why the scenario of providing requirements in tandem with customizing reuse software components as taught by Richburg would necessarily exclude the need to use a mapping as taught by Kim. It is perfectly normal to have legacy of preexisting code in a reuse database; and that does not mean that Richburg by reshuffling the reuse code will render the need of having requirements mapped or traced/validated against proposed implementation or software target deliverables a completely unwanted process. Further, the mapping of functional requirements against what is to be implemented was a known concept in Case tools analysis (refer to section B below). Since Kim happens to use a matrix to implement such a mapping, this, in light of the requirement analysis and Case tools by Richburg, amounts to a solid motivation as to why one ordinary skill in the art would want to use matrix for

Art Unit: 2193

enabling how requirements can be visibly mapped back and fulfilled by the targeted functions thus implemented to meet such requirements, as set forth in the rejection, notwithstanding the fact that Richburg's Case tool analysis can be equated to mapping of requirements against specific or proposed aspect/objects of implementation.

(B) As per arguments for claim 1 and the combination of Richburg/Kim:

The Applicants have submitted that the assertion by the Office Action concerning a known concept of mapping FR to DP is ungrounded; and that support should be provided (Appl. Rmrks, pg. 4-5). Examples of using Use Case to map requirements to components to be assembled to actualize a Use case and traceability mapping are disclosed in:

Visual Modeling with Rational Rose and UML, by Grady Booch, ISBN: 0201310163, April 1998, Addison-Wesley, Ch. 3, and 11;

RequisitePro, Requirements Management, Rational Software Corporation, April 1997, ch. 13-15.

Concerning Applicants' arguments that Kim does not producing software code based on requirements and Richburg does not disclose a set of FR (Appl. Rmrks, pg. 5, middle para), the rejection is a USC 103(a) hence the combination has to be taken as a whole in order to provide basis of obviousness; i.e. in response to applicant's arguments against the references individually, one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). The rejection has shown Richburg's Case tool and requirement gathering; and this in conjunction

Art Unit: 2193

with the mapping of requirements known in Case Tool and the matrix by Kim, has provided basis for the obviousness as set forth in the rejection.

(C) Applicants have submitted that the detailed design document of claim 64 has been implied as being inherent from the Office action, and that corroborative support for it be provided; and that the combination Richburg/Kim fails to disclose such detailed design description of claim 64 (Appl. Rmrks, pg. 6, top 3 para). The rejection has pointed out that the detailed specification being used in conjunction with the requirements-gathering tool by Richburg discloses a form of file storage of specification detail information being persisted in the knowledgebase; and that is the first prong of the rejection. The second prong is providing the rationale as to why it would have been obvious for one ordinary skill in the art to provide Kim's teaching of a matrix so to persist this teaching with the Case tool information and adding this to the knowledgebase by Richburg is presented in the rationale below the implicit teaching of detail specification being stored by Richburg. The case tool storing of detail information and attributes/parameters needed for the Case tools or Use Case analysis has been evidenced in section B above; and based on such persisting of detailed design parameters from the Case tools as well as the knowledgebase file legacy by Richburg, the motivation to provide a detailed information design document to persist the matrix teaching by Kim has been provided in the rejection based on the combination Richburg/Kim as set forth in claim 1 and the rationale in claim 64. There is no implication of any inherent teaching being involved and for which a support is required as asserted by Applicants.

(D) As per arguments of claims 6, 15, 31, 36, 48, 67, and 86, (Appl. Rmrks, pg. 7, second and third para), Applicants seem to come back to the lack of support for the concept of matrix

Art Unit: 2193

mapping in the combination of Richburg's Case tools plus Underwood and Rudolph; but this issue has been raised and addressed in section A-B above. The use of requirement mapping with Case tool or Use Case (including traceability mapping) has been supported with evidence in section B above. The rejection has pointed out how Richburg's use of Case tool can be enhanced with the object-oriented Case tool and requirements mapping features by Underwood; and how a well-known concept like a matrix can be used therefor in conjunction with another approach done by Rudolph which when combined with Richburg or Underwood would also facilitate the selection of the best solution for a given real-world problems in accordance to some user's requirements; such problems can be implemented in software or any other forms.

Concerning the argument that Underwood's use of matrix is not related to design of software; and that Richburg's knowledgebase files are not same as matrix for mapping of requirements (Appl. Rmrks, pg. 7, last para, pg. 8, top), Applicant's arguments against the references taken individually are not persuasive because one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981); *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). As mentioned in section A, Richburg has shown some Case tool mapping; and Case tool validation of requirements has been evidenced in section B. This in conjunction with Underwood's and Rudolph's matrix teaching amount to the motivation as to why one skill in the art in light of teaching of Case tools (implying requirements mapping and traceability features) as taught by Richburg, Rudolf's requirement mapping with matrix; and Underwood's matrix use for validation would want to combine as set forth in the rejection.

Art Unit: 2193

(E) As for the arguments pertaining specifically to claims 6, 15, 31, 36, 48, 67, and 87 (pg. 8-11), Applicants' arguments revolve basically on the alleged inappropriate combination of Richburg, Underwood, and Rudolph. Applicants made no specific point for putting forth or revealing how the rejection has failed to meet the specifics of the claims. The rationale as to why the above combination has been established has laid out the reasons for combining; and Applicants by arguing on the some alleged deficiencies against each reference fail to convince why the rejection should be withdrawn. As for the arguments already addressed, they are referred back to the above sections, accordingly.

Conclusion

8. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (272) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (571)272-3719.


The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence – please consult Examiner before using) or 703-872-9306 (for official correspondence) or redirected to customer service at 571-272-3609.

Art Unit: 2193

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

VAT

May 10, 2005



TODD INGBERG
PRIMARY EXAMINER